An Exact Method for the Double TSP with Multiple Stacks

Larsen, Jesper; Lusby, Richard Martin; Ehrgott, Matthias; Ryan, David

Publication date:
2009

Document Version
Publisher's PDF, also known as Version of record

An Exact Method for the Double TSP with Multiple Stacks

Report 2.2009

DTU Management Engineering

Jesper Larsen
Richard Lusby
Matthias Ehrgott
David Ryan
April 2009

DTU Management Engineering
Department of Management Engineering

An Exact Method for the Double TSP with Multiple Stacks

Richard Lusby[‡,*], Jesper Larsen[‡], Matthias Ehrgott [§], David Ryan [§]

[‡] Department of Management Engineering, Technical University of Denmark,
DK-2800 Kgs. Lyngby, Denmark
[§] Department of Engineering Science, The University of Auckland,
Auckland, New Zealand

February 25, 2009

Abstract

The double travelling salesman problem with multiple stacks (DTSPMS) is a pickup and delivery problem in which all pickups must be completed before any deliveries can be made. The problem originates from a real-life application where a 40 foot container (configured as 3 columns of 11 rows) is used to transport up to 33 pallets from a set of pickup customers to a set of delivery customers. The pickups and deliveries are performed in two separate trips, where each trip starts and ends at a depot and visits a number of customers. The aim of the problem is to produce a stacking plan for the pallets that minimizes the total transportation cost (ignoring the cost of transporting the container between the depots of the two trips) given that the container cannot be repacked at any stage. In this paper we present an exact solution method based on matching *k*-best TSP solutions for each of the separate pickup and delivery TSP problems and show that previously unsolved instances can be solved within seconds using this approach.

Keywords: Routing, packing, TSP, k-best solution, exact method.

1 Introduction

The double travelling salesman problem with multiple stacks (DTSPMS) was originally proposed in [12]. The problem is essentially a pickup and delivery problem where all orders are collected before any delivery takes place. It is based on two graphs each containing n orders. The two graphs define a pickup and a delivery problem each of n pickup or delivery points and the depot. It is not allowed to re-pack the container during the transport. Furthermore, when delivering the orders, only the items "visible" from the end of the container can be taken out. A feasible solution to the problem is defined by a pickup route and a delivery route (both Hamiltonian circuits starting and ending at the depot) and a "stacking plan" for the container. This stacking plan must be feasible with respect to the two routes. The objective function is to minimize the accumulated driving cost in both routes where the cost of going directly from i to j is given by the 2D-Euclidean distance. Figure 1 shows a small example of a solution.

The original problem consists of loading and unloading orders to a 40-foot container. Such a container can be loaded with 33 pallets in 3 columns each containing 11 orders. Any pallet

*Corresponding author. E-mail address: rmlu@man.dtu.dk

picked up can only be placed in one of three places and only three pallets are available for the next delivery at any point (except the last two customers on each route) of the pickup respectively delivery tour.

Extreme cases of the problem occur if there is one column or if there are n columns. Both these cases can be solved by solving the Travelling Salesman Problem (TSP). In the case of one column the delivery must be the reverse of the pickup, so this problem can be solved by summing the arc costs for the pickup and delivery problems. In the case of n columns there are no constraints linking the routes so one simply solves a TSP for the pickup problem and a TSP for the delivery problem.

To the authors' knowledge, the only paper that treats this problem is [12]. Here the problem is generalized to r rows and c columns, and problem instances of sizes 10 (5 rows by 2 columns), 12 (4 rows by 3 columns), 33 (11 rows by 3 columns) and 66 (11 rows by 6 columns) are introduced and solved. Besides giving a mathematical model of the problem, the paper describes the development of two neighbourhoods used in four different metaheuristics (Iterated Local Search (ITS), Tabu Search (TS), Simulated Annealing (SA) and Large Neighborhood Search (LNS)). It is reported that the biggest instances that can be solved based on the mathematical model is of size 12 in a set up with 4 rows and 3 columns. Solution times ranges from 14 to 2850 seconds with an average of 450 seconds. For these instances a running time of 10 seconds is enough for the LNS to find the optimal solution for all 20 instances. SA is typically one to four percent from optimum while TS is a bit worse. With a running time of three minutes the SA is able to solve all but four instances to optimality. For the 33 order instances the LNS is clearly superior to the other metaheuristics. The contribution of [4] adds further neighbourhoods to a Variable Neighbourhood Search.

The DTSPMS can be seen as a one-vehicle pickup and delivery problem with special structure. The problem does, however, differ significantly from the "ordinary" PDP. Firstly, in the DTSPMS all pickups must be performed before any deliveries can be made. Secondly, and perhaps more importantly, one is not allowed to re-stack the container at any stage. This enforces a set of ordering restrictions on the placement of the pallets in the container. PDPs have been surveyed recently in [10, 11].

The TSP with pickup and delivery was first introduced in [6] and an exact method is proposed in [7]. Exact approaches to the regular PDP with one vehicle and a single column based on a LIFO ordering can be found in [1, 3]. Another related problem is the routing problems with backhauls (most closely related to the TSP with backhauls). In backhaul problems all pickups have to be performed before any deliveries can take place. For the single vehicle problem these can be seen as special cases of the more general clustered TSP. An exact approach is presented in [8].

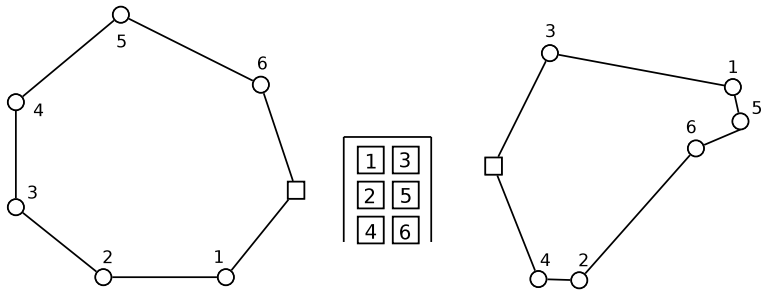

Figure 1: A small example with 6 orders. The pickup and delivery tours are given on the right and left, respectively. Between the two tours is a feasible stacking plan for a container with 2 columns and 3 rows

The rest of the paper is organized as follows. In Section 2 we describe our exact solution

approach for solving the DTSPMS. We begin with a short overview of the method before carefully describing its main components in three subsections. Section 3 presents our experiments and results. We compare our findings with the results of previous work on the same test instances. Finally, Section 4 summarizes our conclusions and gives some directions for future research on this topic.

2 Solution Approach

To solve the DTSPMS to optimality one must find a pickup tour and a delivery tour (henceforth referred to as a pickup and delivery tour combination) that have the shortest combined total distance and which also permit a feasible stacking plan for the pallets in the container. One can observe that the dimensions of the container used to transport the pallets solely dictate whether or not any pickup and delivery tour combination is feasible. As we have already mentioned in the introduction, in the unrealistic case that the container has as many columns as there are customers, the pickup and delivery tour combination consisting of the respective optimal TSP solutions is feasible. Using this observation, we elect to treat the respective pickup and delivery TSP problems separately, preferring to use a solution to each as input to an integer programming formulation, which we call the TSP matching problem, that determines whether or not the order of the customers in the pickup and delivery tour combination allows a feasible stacking plan for the pallets. This a completely different approach to the integrated exact method proposed in [12].

The exact approach we propose simply entails repeatedly matching solutions to two separate TSP problems until a feasible stacking plan is identified. To ensure optimality one must impose certain structure on the order in which solutions to the two TSP problems are considered. That is, the total distance of successive pickup and delivery tour combinations must be at least as bad as the last considered infeasible one. This is achieved by finding the set of k-best solutions to each of the pickup and delivery TSP problem, constructing all pickup and delivery combinations and then ordering the combinations in increasing order of total distance. Algorithms for finding the set of k-best solutions to optimization problems have been proposed in [5] and [9], and more recently reviewed in [13]. By definition, the set of k-best solutions to a TSP problem is the set of k tours where the length of any other tour is at least as long as the longest tour in the set. This paper is, to the authors' knowledge, the first application of finding the k-best solutions to the TSP problem. Note that for any pair of pickup and delivery tours two pickup and delivery tour combinations can be generated. The second combination comes from the fact that the two tours can be traversed in either direction; however, we only need to reverse one to obtain the only different possibility.

In what follows we discuss, in detail, several important components of the algorithm. In particular, Section 2.1 explains the procedure for generating the set of k-best solutions to each of the TSP problems, while Section 2.2 introduces and formulates the integer program used in the TSP matching phase of the algorithm. We conclude with Section 2.3, which provides an overview of the entire algorithm. In outlining the overall approach, Section 2.3 also describes a pre-processing technique that can be used to eliminate pickup and delivery tour combinations from consideration and discusses the termination conditions for the algorithm.

2.1 The set of k-best TSP Solutions

To generate the set of k-best TSP Solutions to each of the pickup and delivery problems we implement the *Lawler Algorithm* (see [9] and [13]). The Lawler algorithm can be identified as a *partitioning algorithm* since at any given iteration the set of feasible solutions to the TSP problem is partitioned into pairwise disjoint sets of tours. This algorithm, like any other approach used to find the set of k-best solutions to an optimization problem, relies on the fact that one has a method

to solve, to optimality, a single instance of the problem. Given the size of the TSP problems we consider is not overly large, we elect to use the 1-tree relaxation branch-and-bound approach to solve any instance of the TSP (see [2]). In implementing this method we utilize the well known Held Karp lower bound and an initial upper bound that is obtained using the nearest neighbour heuristic followed by the 2-opt exchange improvement heuristic.

In order to provide a more formal description of the Lawler Algorithm, we introduce the following notation and definitions. Let $G = (V, E)$ be a graph with vertex set $V = \{1, 2, \ldots, n\}$ and edge set $E = \{e_1, e_2, \ldots, e_m\} \subseteq \{(i, j) : i, j \in V, i \neq j\}$. We denote the vector of edge lengths $D = [d(e_1), d(e_2), \ldots, d(e_m)]^T \in \mathbb{R}^m$, where $d(e)$ is the length of e. The pair (G, D) will be termed a *weighted graph*. We assume without loss of generality that G is Hamiltonian, and denote the set of all tours by $\mathscr{H}$. The length of any tour $H \in \mathscr{H}$ is given by $L(H) := \sum_{e \in H} d(e)$. Let $1 \leq k \leq |\mathscr{H}|$ and define $\mathscr{H}(0) := \emptyset$. Any set $\mathscr{H}(k) = \{H_1, H_2, \ldots, H_k\} \subseteq \mathscr{H}$ satisfying $L(H_1) \leq L(H_2) \leq \cdots \leq L(H_k) \leq L(H)$ for all $H \in \mathscr{H} \backslash \mathscr{H}(k)$ is termed a set of k-best tours. To complete the definitions we introduce the set $T_{I,O} = \{H \in \mathscr{H} : I \subseteq H \subseteq E \backslash O\}$, which is known as a *restricted set of tours* and is defined for any $I, O \subseteq E$, where $I \cap O = \emptyset$. I is the set of edges that must be included in the tour, and O is the set of edges that cannot be part of a tour in the *restricted set of tours*.

The k-best TSP is the problem of finding $\mathscr{H}(k)$ in (G, D). At iteration i of the Lawler algorithm $\mathscr{H}(i-1)$ has been determined and one constructs a set of candidates λ for the ith best solution by partitioning $\mathscr{H} \backslash \mathscr{H}(i-1)$ into a set of restricted sets of tours $S_i = \{T_{I_j, O_j}\}$ of cardinality N_s, where $j = 1, \ldots, N_s$. Each $\lambda_j \in \lambda$ is the best solution in the corresponding set T_{I_j, O_j}, and the ith best solution, H_i, is hence the best solution in λ. The i-best set of TSP solutions is given by $\mathscr{H}(i-1) \cup H_i$. Note that any restricted instance of the TSP can be solved as a non-restricted instance by modifying the edge lengths appropriately. One sets the length of any edge $e \in I_j$ to $-M$ (where M is a large positive number), while any edge $e \in O_j$ has its length set to ∞ (see [5] for details). On finding solution H_i, it is removed from the set of feasible solutions, $\mathscr{H} \backslash \mathscr{H}(i)$, for the subsequent iteration (if necessary) by partitioning the restricted set of tours $T^*_{I_j, O_j}$, which it was an element of, even further. This partitioning step entails constructing the set $T^*_{I_j, O_j} \backslash H_i$ and is achieved by creating the further restricted sets of tours $\{H \in T^*_{I_j, O_j} : \{e_1, e_2, \ldots, e_{j-1}\} \subseteq H \subseteq E \backslash \{e_j\}\}$ for $j = 1, 2, \ldots, n$, where $H_i = \{e_1, e_2, \ldots, e_n\}$, and $\{e_1, e_2, \ldots, e_{j-1}\} = \emptyset$ for $j = 1$. Algorithm 1 gives an overview of the entire algorithm.

Algorithm 1: The Lawler Algorithm for determining k-best TSP solutions

Input : (G, D) and some integer $1 \leq k \leq |\mathscr{H}|$.
Output: A set of $\mathscr{H}(k)$ of k-best tours in (G, D).

```
S_1 := {H} = {T_{I_1 O_1} : I_1 = ∅, O_1 = ∅}; H(0) := ∅;
for i ← 1 to k do
    for j ← 1 to N_s do
        Identify the best tour λ_j in each {T_{I_j O_j}} ;
    Construct the set of candidate solutions λ;
    Identify H_i;
    Set H(i) = H(i − 1) ∪ H_i;
    Set S_{i+1} = {S_i \ {T*_{I_j O_j}}};
    for j ← 1 to n do
        S_{i+1} = S_{i+1} ∪ {H ∈ T*_{I_j,O_j} : {e_1, e_2, ..., e_{j−1}} ⊆ H ⊆ E \ {e_j}} ;
```

Algorithm 1 assumes that (G, D) has at least k tours. We have chosen the Lawler Algorithm in preference to the alternative approach of determining k-best tours presented in [5] since it has

a better worst case performance on the number of TSPs that need to be solved to determine the k-best set. We build this algorithm into a dynamic framework since we do not know prior to solving the problem the exact value of k we will need to prove optimality. This is discussed in Section 2.3.

2.2 The TSP Matching Problem

At the core of our approach is an integer program that we have termed the TSP matching problem. This problem receives as input a pickup tour and a delivery tour and determines whether or not a feasible stacking plan for the pallets in the container exists. The solution to this problem, if it exists, must identify a row and column position for each pallet and satisfy a number of sequencing constraints defined by the input tours. Here we provide a formal description of the model.

Let us assume that we have a container which can hold n pallets and which also has dimensions $c \times r$, where c denotes the number of columns and r denotes the number of rows. Note that $n = c \times r$ and all TSP tours have $n + 1$ vertices (n customers plus the depot). We introduce the bijective functions $\pi : \mathbb{Z} \rightarrow \mathbb{Z}$ and $\delta : \mathbb{Z} \rightarrow \mathbb{Z}$ which provide the pickup and delivery number for the pallet numbers, respectively. For example, $\pi(1) = 5$ states that pallet number one will be picked up 5th. Given the aforementioned definitions, the placement of a pallet in a particular position in the container is modelled by the following binary decision variable:

$$x_{ij}^p = \begin{cases} 1 & \text{if pallet } p \text{ is placed in position } (i,j) \\ 0 & \text{otherwise} \end{cases}$$

where $i = 1, \ldots, r$ denotes a row position (ordered from the back of the container), while $j = 1, \ldots, c$ denotes a column position.

Our aim is to find the shortest pickup and delivery tour combination that permits a feasible stacking plan for the pallets. The TSP matching problem is simply a tool that tests for such feasibility, and as such one has flexibility in the choice of an objective function. We define the following objective function, which does not discriminate among the set of feasible solutions and will return a value of n if the pickup and delivery tour combination are compatible.

$$Z = \text{ Min } \sum_{i}^{r} \sum_{j}^{c} \sum_{p}^{n} x_{ij}^p$$

There are four sets of constraints the decision variables are required to satisfy. The first two, (1) and (2) below, are typical assignment constraints that stipulate each pallet must occupy one position, and that each position is occupied by only one pallet.

$$\sum_{i}^{r} \sum_{j}^{c} x_{ij}^p = 1 \text{ for all } p \tag{1}$$

$$\sum_{p}^{n} x_{ij}^p = 1 \text{ for all } i, j \tag{2}$$

The last two use the order of the customers in each of the pickup and delivery tours and enforce pickup and delivery sequencing constraints. Each sequencing constraint pertains to adjacent positions in the same column and ensures pallets are in the correct sequence from both a loading as well as an unloading perspective. Let us first consider the delivery tour. If pallet p is to be placed in position (i, j), then the pallet p' immediately in front of it, i.e. the pallet in position $(i + 1, j)$, must be delivered earlier in the route (except for row r of course). That is, $\delta(p') < \delta(p)$. Otherwise, pallet p would not be able to be accessed. Hence, we get the following set of constraints for each pallet $p = 1, \ldots, n$. Here, p' indexes the set of pallets $\delta' = \{i : 1 \leq i \leq n, \delta(i) < \delta(p)\}$.

$$x_{ij}^{p} \leq \sum_{p' \in \delta'} x_{i+1,j}^{p'} \text{ for all feasible positions } (i,j) \text{ of } p \tag{3}$$

The pickup tour leads to a set of similar constraints. This time, if pallet p is to be loaded into position (i,j), then the position immediately in front of it, i.e. $(i-1,j)$ must be occupied by a pallet p' that has already been loaded (except for row 1 of course). That is, $\pi(p') < \pi(p)$. Hence, the following set of constraints for each pallet $p = 1,\ldots,n$. Here, p' indexes the set of pallets $\pi' = \{i : 1 \leq i \leq n, \pi(i) < \pi(p)\}$.

$$x_{ij}^{p} \leq \sum_{p' \in \pi'} x_{i-1,j}^{p'} \quad \text{for all feasible positions } (i,j) \text{ of } p \tag{4}$$

Note that not all pallets can be placed in all positions. The set of available positions for a pallet is dependent on when the pallet is picked up and when the pallet is delivered. For example, if $\pi(i) = 1$ then pallet i can only be placed in a position in the first row. Similarly, if $\delta(i) = 2$, then pallet i can only be placed in some position in rows $r-1$ and row r. Hence, constraint (3) and (4) are only defined for the feasible positions for each pallet p based on $\pi(p)$ and $\delta(p)$. The integer program has $r \times c \times n$ binary variables in total. There are n constraints of the form (1), and $r \times c$ constraints of the form (2). It is somewhat difficult to give a precise statement on the number of constraints of the form (3) and (4). We note that there are the same number of each in total; however, the exact number is dependent on the configuration of the container. One can see that if $\pi(i) < r$, then i has $c(\pi(i)-1)$ pickup sequencing constraints. If, on the other hand, $\pi(i) > n - r + 1$, then pallet i will have $c(n - \pi(i) + 1)$ such constraints. In all other cases, each pallet will have $c(r-1)$ pickup sequencing constraints.

2.3 The Complete Algorithm

Given the discussion in Sections 2.2 and 2.1, we are now in a position to provide a formal description of the entire solution approach. The method begins with the construction of the set of 20 best tours for each of the respective pickup and delivery tour problems. The value of 20 has been chosen since one cannot discount the possibility that near optimal solutions to each of the TSP problems provide a compatible pickup and delivery tour combination. By restricting the initial set of best tours one can, possibly, prevent the needless generation of much bigger sets. All possible pickup and delivery tour combinations are generated from these sets (remembering to generate two possible combinations for each pair of pickup and delivery tours) and ordered by increasing total distance. The TSP matching problem is then run for each combination. If no feasible pickup and delivery tour combination is found, or additional tours are required to prove optimality, we generate the next set of 30 best tours. Again, there is no important reason for incrementing by 30. Several parameters were tried, and this combination (20 tours initially and increments of 30) seemed to work well. This process is then repeated until the optimal solution is found.

On finding a feasible solution, we must continue to generate subsequent sets of 30 best tours until two conditions are satisfied. Firstly, the incumbent solution must be shorter than, or at least equal to, the pickup and delivery tour combination consisting of the optimal pickup tour and the longest delivery tour already generated. The optimal delivery tour and the longest pickup tour already generated impose a similar restriction. In any case, these two combinations provide a lower bound on the the length of any pickup and delivery tour combination not contained in the list generated from the respective k-best sets. We refer to these as the BPWD (best pickup worst delivery) bound and the BDWP (best delivery worst pickup) bound. A lower bound on any pickup and delivery tour combination not yet generated is the minimum of the BPWD bound and the BDWP bound. Once this lower bound is greater than or equal to our incumbent solution, we

know that no pickup and delivery tour combination can have a shorter length. Hence the algorithm terminates. While solving a problem it may happen that one of the bounds exceeds the length of the incumbent solution. In this case generation of subsequent tours for the relevant TSP problem stops.

Algorithm 2: The Complete Algorithm

```
   Input  : Two weighted graphs and one container.
   Output: An optimal stacking plan for the container.
1  Generate 20 best pickup tours and delivery tours;
2  Construct pickup and delivery tour combinations;
3  while Not solved do
4      Identify lower bound;
5      Solve TSP matching;
6      if Feasible stacking plan found then
7          if lower bound ≥ incumbent then
8              Solved true;
9          else
10             Incumbent = stacking plan;
11             if BPWD bound < incumbent then
12                 Generate next set of 30 best delivery tours;
13             if BDWP bound < incumbent then
14                 Generate next set of 30 best pickup tours;
15             Construct pickup and delivery tour combinations;
16     else
17         Generate next set of 30 best pickup and delivery tours;
18         Construct pickup and delivery tour combinations;
```

To conclude this section we consider the pickup and delivery tour combination construction part of the algorithm (see lines 2, 15, and 18) in slightly more detail. Here one would like to construct a TSP matching problem only if one is fairly confident a feasible stacking plan exits. By implementing a pre-processing algorithm that attempts to identify obviously infeasible matchings, one can reduce the number of TSP matching problems that must be solved. By considering the order in which certain pallets will be picked up and delivered (as specified by the respective tours) one can perform simple comparisons of $\pi(p)$ and $\delta(p)$ for each pallet p to detect infeasibility. For example, it is trivial to see that a pallet that is picked up first cannot be delivered first. This can be generalized to any pallet p with $\pi(p) \leq r$ must have $\delta(p) \geq r - \pi(p) + 1$. Also, any pallet p with $\pi(p) \geq n - (c-1)r + 1$ must have $\delta(p) \leq n - \pi(p) + (c-1)r + 1$. Only tours which satisfy such restrictions are used to create a TSP patching problem.

3 Results

To allow accurate comparisons with previous work to be made, the proposed methodology has been tested on a number of instances constructed from the data sets used in [12]. These data sets have 33 customers randomly positioned on a 100×100 grid. The coordinates of the lower left-hand corner of this grid are (0.0, 0.0), while the upper right-hand corner has coordinates (100.0, 100.0). The depot is assumed to be positioned at the point (50.0, 50.0). Instances with fewer than

33 customers are constructed by taking the first so many customers from each data set. The cost on any arc is the 2D-Euclidean distance rounded to the nearest integer.

According to [12], the largest instances that could be solved to optimality within an hour of computing time using the authors' exact method were 12 customers with a container configuration of 4×3 and 10 customers with a container configuration of 5×2, respectively. While the authors do not mention exact running times for the latter instances, solution times for the former instances range from 14 seconds to 2850 seconds and have an average running time of 450 seconds. All tests were performed on a 1.6 GHz Dell laptop with 1.5GB RAM. Table 1 and Table 2 give the results of the same test instances using our k-best TSP approach. Within each of the tables, the following information is provided: the test case ID, the value of the best solution obtained (Z^*), the value of k for both the pickup and the delivery tour (k_P and k_D) that give this solution, the lower bound L at termination, and the bound gap between Z^* and L. We also report the number of potentially feasible TSP matching problems identified (T), the number of TSP matching problems actually solved (T_S), the computation time in seconds (t), and the cardinality of the k-best set at termination for each of the pickup and delivery tours (given by N_P and N_D, respectively). For example, test case R00 in Table 1 has an optimal solution of 694. This solution uses the optimal pickup tour and the 25th best delivery tour. We solved 59 out of a possible 848 TSP matching problems, and the algorithm took 4 seconds. At termination of the algorithm the cardinality of each of the k-best sets was 50, and the lower bound on the length of any pickup and delivery tour combination not yet considered was 698. The fact that this bound is higher than the optimal solution of 694 is evidence that we have generated more tours in our k-best sets than is necessary. The entire algorithm has been written using the C++ programming language and utilizes the ILOG Concert Technology API to implement the TSP matching problem in Cplex 10.0. All tests have been performed on an operating system running Suse 10.1 with a dual core AMD 2.2 GHz processor and 2GB of RAM. A maximum running time of three hours is also enforced.

Table 1: 12 Customers – 4×3 Container

Case	Z^*	k_P	k_D	L	%	T	T_S	t	N_P	N_D
R00	694	1	25	698	0.00	848	59	4	50	50
R01	710	2	1	716	0.00	212	1	3	20	20
R02	606	6	5	606	0.00	198	32	1	20	20
R03	680	7	1	686	0.00	259	9	2	20	20
R04	607	1	7	611	0.00	215	5	1	20	20
R05	567	9	1	573	0.00	169	5	1	20	20
R06	747	21	1	754	0.00	593	37	2	50	20
R07	557	65	1	558	0.00	3201	229	5	80	50
R08	690	6	1	697	0.00	228	3	2	20	20
R09	669	1	2	674	0.00	258	6	1	20	20
R10	633	2	448	633	0.00	8717	255	13	50	470
R11	591	37	4	592	0.00	1386	214	7	50	80
R12	722	1	4	731	0.00	128	3	1	20	20
R13	664	13	2	665	0.00	106	7	1	20	20
R14	650	116	1	653	0.00	5738	1848	20	140	80
R15	595	25	1	598	0.00	877	130	4	50	50
R16	577	1	37	578	0.00	477	39	3	20	50
R17	737	18	2	737	0.00	132	15	2	20	20
R18	724	4	8	725	0.00	627	83	5	50	20
R19	753	11	1	756	0.00	265	29	2	20	20

The results in Table 1 highlight the efficiency of the proposed methodology. One can observe a dramatic improvement in the solution times compared to those reported in [12]. All instances can be solved to optimality within 20 seconds, with an average running time of 4 seconds. A

feasible pickup and delivery tour combination is often found quickly and it is interesting to note that, more often than not, the optimal tour to one of the TSP problems appears in the optimal solution. Furthermore, the cardinality of the sets of k-best tours is almost always small and one can observe that only a small number of TSP matching problems are required to be solved. The results suggest that the 12 customer case and a 4×3 container is trivial.

Table 2 reports similar findings for the 10 customer and a 5×2 container. Here all instances are solved within 114 seconds, with an average running time of 9.5 seconds. We do, however, observe an increase in the number of TSP matching problems that must be solved and larger $k-$best sets required to prove optimality. This is not too surprising since, even though it has fewer pallets, from a stacking perspective it is less flexible than the 4×3 case. In other words, one may have to generate more tours before a feasible pickup and delivery combination can be found.

Table 2: 11 Customers – 5×2 Container

Case	Z^*	k_P	k_D	L	%	T	T_S	t	N_P	N_D
R00	680	1	186	681	0.00	1176	154	5	80	200
R01	704	1	107	704	0.00	1231	299	3	110	110
R02	629	2	98	629	0.00	3199	183	6	290	140
R03	610	4	13	610	0.00	10	3	1	20	20
R04	614	16	5	618	0.00	306	4	3	80	80
R05	546	2	556	548	0.00	1279	296	4	110	80
R06	774	1	161	775	0.00	4376	308	5	170	170
R07	547	91	2	549	0.00	1225	29	1	110	80
R08	670	19	15	670	0.00	2476	101	5	410	110
R09	610	42	1	610	0.00	143	27	1	50	20
R10	624	2	190	624	0.00	5067	854	7	200	200
R11	536	1	50	536	0.00	692	74	2	50	50
R12	678	1	56	681	0.00	521	151	3	80	80
R13	654	18	6	654	0.00	274	21	2	110	50
R14	603	50	32	603	0.00	6827	352	13	860	140
R15	586	16	9	587	0.00	230	22	2	110	50
R16	535	7	412	535	0.00	18595	620	114	80	3470
R17	729	5	155	729	0.00	4054	1497	11	230	170
R18	616	6	9	616	0.00	29	2	1	20	50
R19	650	24	1	654	0.00	429	34	1	50	50

Given the excellent results on what were previously considered difficult instances, we decided to construct four more test cases with 12, 14, 15, and 18 customers in order to ascertain the limit of the current approach. Table 3 gives the results for 20 instances of 12 customers with a 6×2 container. While some instances can be solved within seconds using this approach (in particular R03, R12, and R18), one can observe that there is generally a significant increase in the computational effort required to prove optimality. Results indicate that much bigger k-best sets are required (more than 18,000 for the pickup tour in instance R08) and many more TSP matching problems must be solved (more than 72,000 for R05). It is interesting to see that in some cases k_P and k_D can also be quite large. For example, the optimal solution of instance R17 in Table 3 contains the 6531st best delivery tour. Furthermore, one instance could not be solved to optimality within 3 hours; however it is extremely close, with a bound gap of 0.13%

Table 4 presents the results for 20 test instances with 14 customers and a 7×2 container. The results again show an escalation in the computational effort required to prove optimality, with very few instances being solved to optimality within 3 hours of computation time. The 19 instances that return a feasible solution are within, on average, 1.8% of optimality. One instance, R14, did not return a feasible solution within 3 hours. Looking at the statistics for this instance, it is easy to explain why. The algorithm evaluated 478,981 infeasible TSP matching problems. This

Table 3: 13 Customers – 6×2 Container

Case	Z^*	k_P	k_D	L	%	T	T_S	t	N_P	N_D
R00	726	21	236	726	0.00	25612	629	142	560	1970
R01	741	5	134	741	0.00	2033	180	17	350	200
R02	660	34	684	660	0.00	474682	2904	2432	9500	2120
R03	690	28	1	691	0.00	206	24	4	50	50
R04	659	20	1526	659	0.00	260583	11110	1151	1550	7070
R05	631	4	6385	631	0.00	3067971	72573	2126	8000	6800
R06	793	5	409	793	0.00	194332	6099	310	3140	680
R07	593	2898	2	593	0.00	562686	10802	485	3860	1820
R08	749	372	47	748	0.13	811333	3431	10823	18320	1730
R09	692	14	23	692	0.00	12937	675	45	1040	80
R10	663	27	322	663	0.00	219767	8369	2452	440	13970
R11	625	6	1445	626	0.00	220608	2022	356	1460	3320
R12	741	1	84	742	0.00	396	184	7	110	110
R13	694	90	9	694	0.00	5309	311	16	410	230
R14	680	139	58	680	0.00	48065	996	205	3410	350
R15	628	324	39	628	0.00	100600	1720	306	2720	2600
R16	610	7	2601	610	0.00	163042	3490	3537	320	11450
R17	780	8	6531	780	0.00	1173533	39030	3832	5960	10070
R18	736	1	105	736	0.00	1253	287	16	260	110
R19	789	97	75	789	0.00	25311	1447	171	2210	350

is significantly more than any other test instance. A more sophisticated infeasibility check may have been able to more efficiently rule out the need to solve the majority of these. Again the table is characterized by large k-best sets and many potential TSP matching problems.

As a comparison, Table 5 gives the results for 20 random test instances with 15 customers and a 5×3 container. Here one observes that all instances are solved to optimality within 6172 seconds, with an average running time of 492 seconds. In contrast to Table 4, the k-best sets are significantly smaller and many fewer TSP matching problems are solved. This reinforces the fact that it is not the number of customers that determines the complexity of this problem, but rather the dimensions of the container. We also experimented with 18 customers and a 6×3 container. We have omitted a table of the results, but will provide a brief summary of the results. For the 20 test instances, only 5 could be solved to optimality within 3 hours. Of the 17 that returned feasible solutions, the average bound gap was, however, only 0.90%. The three instances that did not return a feasible solution were very similar to the only other such instance discussed earlier. That is, all 3 instances required the evaluation of more than 440,000 infeasible TSP matching problems. On the other hand, one instance did take just 58 seconds to solve. This highlights the fact that one can not discount the possibility that only small k-best sets for the pickup and delivery tours will be required. However, given the increase in the number of instances for which a feasible solution could not be found, without better infeasibility checks for the TSP matching problems, this is where we feel the limit for the current methodology is.

4 Conclusions and Future Work

In this paper we have considered the double travelling salesman problem with multiple stacks and presented an exact solution approach that entails repeatedly matching k-best solutions to the respective pickup and delivery tours. This is to the our knowledge, the first application of finding the k-best solutions to the TSP problem. We have shown that this approach significantly outperforms the only other exact method proposed for this problem and, in doing so, pushed the boundary on what is now solvable within 3 hours of computing time. The structured way in which the k-best

Table 4: 14 Customers – 7×2 Container

Case	Z^*	k_P	k_D	L	%	T	T_S	t	N_P	N_D
R00	774	391	590	765	1.18	3094067	6728	10838	11870	11870
R01	761	47	894	761	0.00	188883	3313	653	2150	4010
R02	690	6021	29	680	1.47	564895	108553	10863	16490	6050
R03	791	18	3708	790	0.13	748748	34425	10807	4610	16880
R04	757	3899	1869	711	6.47	718974	229719	10805	8900	8900
R05	775	141	4542	760	1.98	3622953	113693	10807	12530	12530
R06	824	966	97	818	0.73	6241825	44167	10814	13940	7910
R07	697	6265	14	682	2.20	1653270	9546	10822	11660	11180
R08	831	8105	359	786	5.73	1485526	98459	10815	12320	12320
R09	739	19	78	739	0.00	22191	1847	211	1940	170
R10	733	2135	227	718	2.09	2768820	95904	10845	14090	14090
R11	725	2955	170	705	2.84	2220192	128535	10815	11480	11480
R12	803	28	4905	795	1.01	1249651	25005	10814	9410	14120
R13	746	99	592	746	0.00	413917	4608	1499	3320	7100
R14	–	–	–	710	–	484144	478981	10841	7910	7910
R15	765	48	327	765	0.00	89373	1089	1152	2840	3470
R16	685	228	171	679	0.88	788267	13770	10826	1280	15380
R17	828	6477	1572	793	4.41	601991	206365	10860	10340	10340
R18	774	6826	1	774	0.00	904811	37346	3607	6830	6830
R19	843	120	3702	828	1.81	2077836	180361	10839	11990	11990

Table 5: 15 Customers – 5×3 Container

Case	Z^*	k_P	k_D	L	%	T	T_S	t	N_P	N_D
R00	741	2	439	741	0.00	72842	7560	207	530	650
R01	754	1	976	754	0.00	270874	34308	587	800	980
R02	658	226	7	658	0.00	40825	5874	167	740	200
R03	768	4	567	768	0.00	80790	3430	160	590	770
R04	708	1	300	708	0.00	18811	4054	103	260	320
R05	737	5	82	737	0.00	13630	949	56	560	110
R06	836	53	14	836	0.00	6284	562	47	6410	80
R07	690	137	23	690	0.00	205211	7407	415	300	260
R08	826	495	1	826	0.00	35363	1320	73	500	260
R09	768	49	9	768	0.00	5218	691	29	260	50
R10	698	274	1	698	0.00	84084	5076	360	290	1160
R11	684	7	87	684	0.00	27501	1748	82	500	230
R12	751	2	57	752	0.00	2807	788	21	80	110
R13	744	188	2	744	0.00	14610	1699	150	200	380
R14	751	5644	2	751	0.00	7661036	113683	6172	9620	3080
R15	748	239	1	748	0.00	1415	3964	92	260	260
R16	692	227	1	692	0.00	157285	14907	652	230	2390
R17	783	268	1	783	0.00	72234	11071	307	290	1250
R18	783	1	315	783	0.00	37667	4915	133	320	320
R19	800	59	14	800	0.00	3724	1250	30	110	80

sets construct the pickup and delivery tour combinations typically ensures that the first feasible solution is often of high quality. Proving optimality, however, may take some time. We have also shown that it is not the number of customers in the problem that creates the complexity, but rather the dimensions of the container that is used.

The results indicate that the 33 customer instances are unlikely to be solvable within 3 hours without further refinement in the proposed methodology. Some possible directions for future work include the following. Firstly, as has been mentioned earlier, identifying obviously infeasible TSP matching problems is of paramount importance to the speed of this algorithm. Hence, improvements in detecting infeasible pickup and delivery combinations without having to construct a TSP matching problem is the direction with the highest priority. Secondly, the list of TSP matching problems that must be solved is a process that can be implemented in a parallel computing environment. Further work in such directions should make it possible to solve bigger instances to optimality within reasonable time.

References

[1] F. Carrabs, R. Cerulli, and J.-F. Cordeau. An additive branch-and-bound algorithm for the pickup and delivery traveling salesman problemwith lifo or fifo. *INFOR*, 2006.

[2] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver. *Combinatorial Optimization*. Wiley-Interscience, New York, 1998.

[3] J.-F. Cordeau, M. Iori, G. Laporte, and J. Salazar-González. A branch-and-cut algorithm for then pickup and delivery travelins salesman problem with lifo loading. *Networks*, (to appear).

[4] A. Felipe, M. Ortuño, and G. Tirado. Neighborhood structures to solve the double tsp with multiple stacks using local search. In *Proceedings of the 8th International FLINS Conference*, World Scientific Proceedings Series on Computer Engineering and Information Science, pages 701 – 706, 2008.

[5] H. W. Hamacher and M. Queyranne. *K* best solutions to combinatorial optimization problems. *Annals of Operational Research*, 4:123–143, 1985.

[6] H. Hernández-Pérez and J. J. Salazar-González. The one-commodity pickup-and-delivery travelling salesman problem. In M. Jünger, G. Reinelt, and G. Rinaldi, editors, *Combinatorial Optimization-Eureka, You Shrink!*, volume 2570 of *Lecture Notes in Computer Science*, pages 89 – 104. Springer, 2003.

[7] H. Hernández-Pérez and J. J. Salazar-González. A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. *Discrete Applied Mathematics*, 145:126 – 139, 2004.

[8] K. Jongens and T. Volgenant. The symmetric clustered traveling salesman problem. *European Journal of Operational Research*, 19:68 – 75, 1985.

[9] E. Lawler. A procedure for computing the *K* best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science*, 18(7):401–405, 1972.

[10] S. N. Parragh, K. F. Doerner, and R. F. Hartl. A survey on pickup and delivery problems. part i: Transportation between customers and depot. *Journal für Betriebswirtschaft*, 58:21 – 51, 2008.

[11] S. N. Parragh, K. F. Doerner, and R. F. Hartl. A survey on pickup and delivery problems. part ii: Transportation between pickup and delivery locations. *Journal für Betriebswirtschaft*, 58:81 – 117, 2008.

[12] H. L. Petersen and O. B. G. Madsen. The double travelling salesman problem with multiple stacks – formulation and heuristic solution approaches. *European Journal of Operational Research*, 2008. Forthcoming, doi:10.1016/j.ejor.2008.08.009.

[13] E. S. van der Poort, M. Libura, G. Sierksma, and J. A. A. van der Veen. Solving the *k*–best traveling salesman problem. *Computers and Operations Research*, 26:409 – 425, 1998.

The double travelling salesman problem with multiple stacks (DTSPMS) is a pickup and delivery problem in which all pickups must be completed before any deliveries can be made. The problem originates from a real-life application where a 40 foot container (configured as 3 columns of 11 rows) is used to transport up to 33 pallets from a set of pickup customers to a set of delivery customers. The pickups and deliveries are performed in two separate trips, where each trip starts and ends at a depot and visits a number of customers. The aim of the problem is to produce a stacking plan for the pallets that minimizes the total transportation cost (ignoring the cost of transporting the container between the depots of the two trips) given that the container cannot be repacked at any stage. In this paper we present an exact solution method based on matching k-best TSP solutions for each of the separate pickup and delivery TSP problems and show that previously unsolved instances can be solved within seconds using this approach.

ISBN 978-87-90855-37-6

DTU Management Engineering
Department of Management Engineering
Technical University of Denmark

Produktionstorvet
Building 424
2800 Kongens Lyngby
Tel. 45 25 48 00
Fax 45 93 34 35

www.man.dtu.dk